2023 Quantum Fair

Solve Shortest Path Problem by QAOA

고려대학교 정성훈
고려대학교 김성훈
고려대학교 정원중

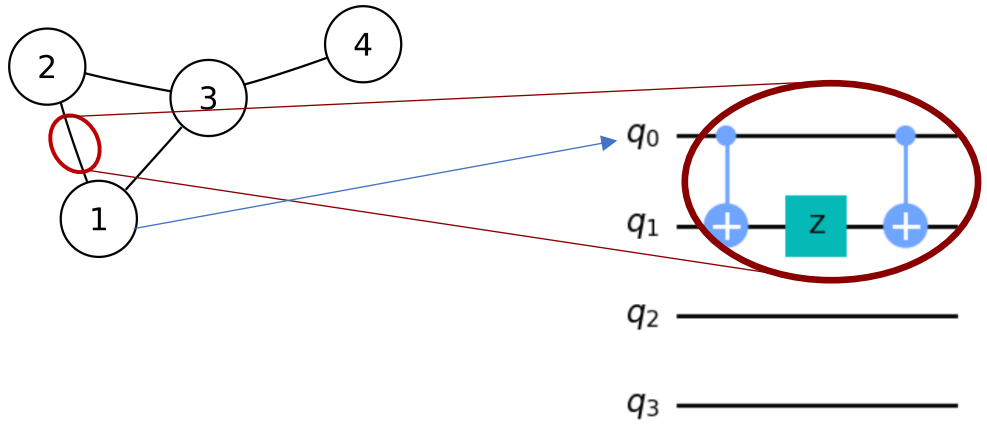
프로젝트 목적

방향 가중 그래프(Directed Weighted Graph)에 대해 시작 Node와 목적지 Node가 주어졌을 때 목적지 Node까지의 최단 경로를 찾는 알고리즘을 QAOA로 구현해보고 간단한 경우에 대한 실행을 확인한다.

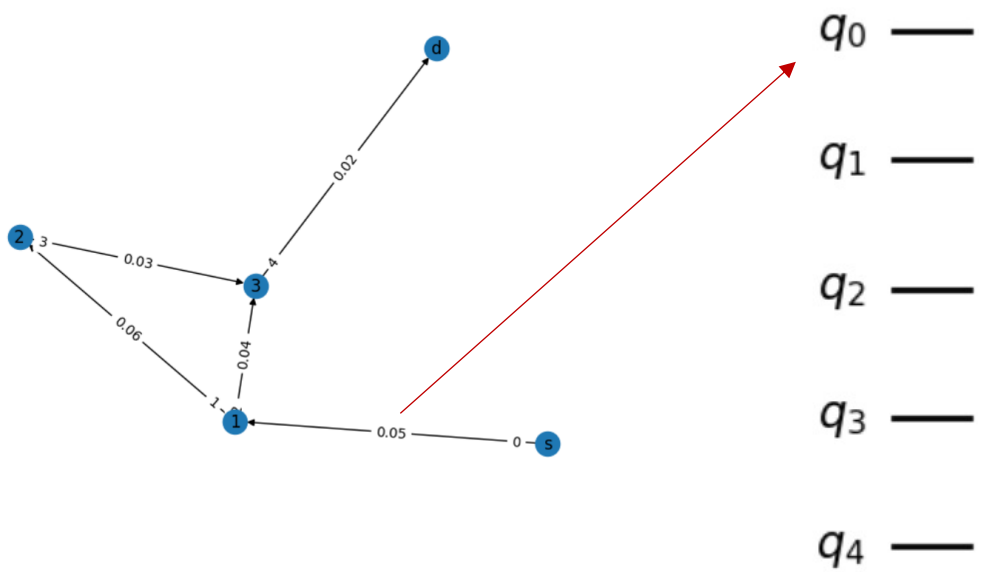
Max-Cut Problem의 QAOA와 유사하게 Adiabatic theorem을 적용하여 Circuit을 설계한다.

Max-Cut Problem & SPP

- Max-Cut Problem
  - Embed Node to qubit
  - Express Edge as ZZ Gate (Express Undirected graph use quantum circuit)

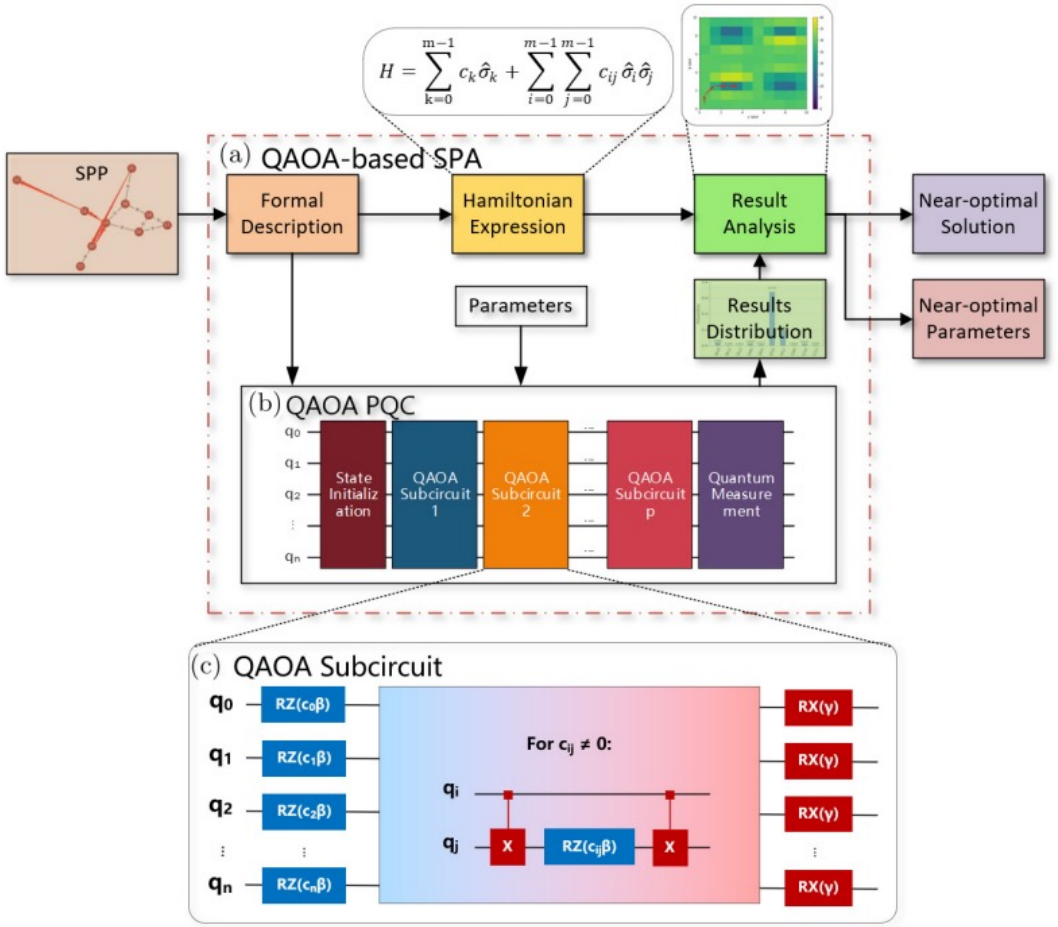


- SPP
  - Embed Edge to qubit



  - How to Express Graph Structure?

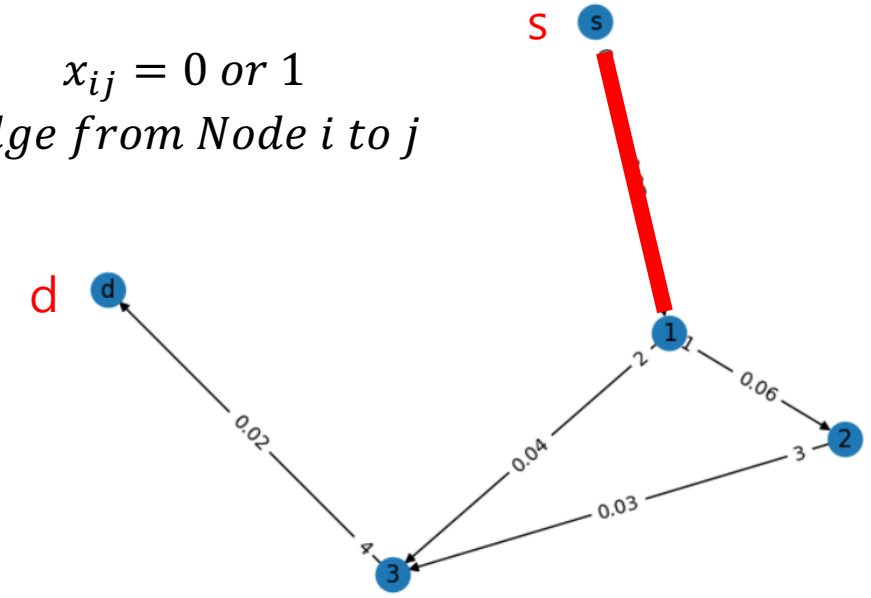
Shortest Path Problem by QAOA



Algorithm Designing

$x_{ij} = 0 \; or \; 1$

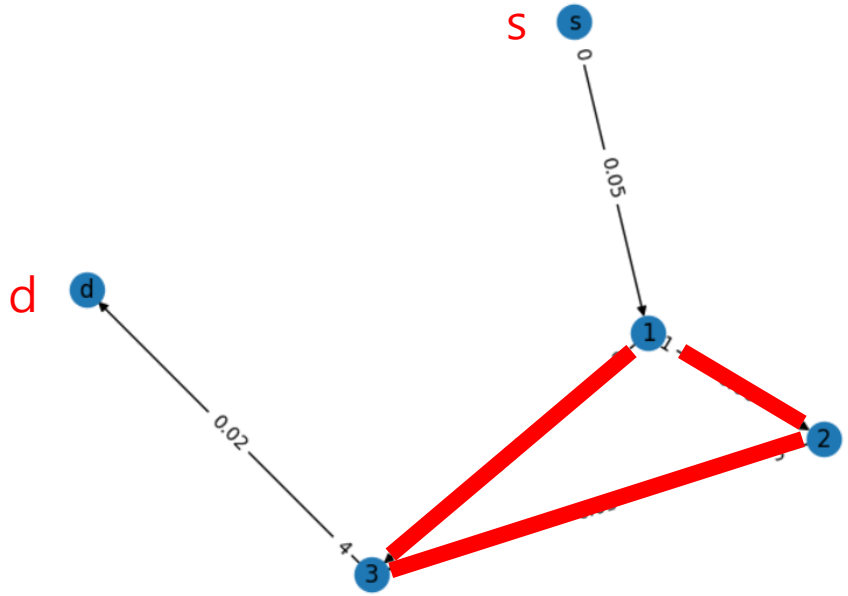
$Edge \; from \; Node \; i \; to \; j$



$$\sum_{j=1}^{n} x_{sj} = 1$$

$$\sum_{i=1}^{n} x_{id} = 1$$

Algorithm Designing



$$\sum_{i=1}^{n} x_{ik} = \sum_{j=1}^{n} x_{kj}$$

Objective Function

$$C = \sum_{i,j} w_{ij} x_{ij}$$

$$F = C + M\left(\sum_{j=1}^{n} x_{sj} - 1\right)^2 + M\left(\sum_{i=1}^{n} x_{id} - 1\right)^2 + M \sum_{k \in V, k \neq s,d} \left(\sum_{i=1}^{n} x_{ik} - \sum_{j=1}^{n} x_{kj}\right)^2$$

Hamiltonian Construction

$$F = C + M\left(\sum_{j=1}^{n} x_{sj} - 1\right)^2 + M\left(\sum_{i=1}^{n} x_{id} - 1\right)^2 + M \sum_{k \in V, k \neq s,d} \left(\sum_{i=1}^{n} x_{ik} - \sum_{j=1}^{n} x_{kj}\right)^2$$

$$F = \sum_{i.j} w_{ij} x_{ij} + M\left\{\left(\sum_{j=1}^{n} x_{sj} - 1\right)^2 + \left(\sum_{i=1}^{n} x_{id} - 1\right)^2 + \sum_{k \in V, k \neq s,d} \left(\sum_{i=1}^{n} x_{ik} - \sum_{j=1}^{n} x_{kj}\right)^2\right\}$$

$$= \sum_{i.j} w_{ij} x_{ij} + M\left\{\sum_{j=1}^{n}\sum_{i=1}^{n} x_{sj}^2 + c_{ij_0} x_{si} x_{ji} + c_0 + \sum_{j=1}^{n}\sum_{i=1}^{n} x_{id}^2 + c_{ij_1} x_{id} x_{id} + c_1 + \sum_{k \in V, k \neq s,d} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{jk} x_{kj}^2 + c_{ik} x_{ik}^2 + c_{ij_2} x_{kj} x_{ik}\right\}$$

Hamiltonian Construction

$$\sum_{i.j} w_{ij}x_{ij} + M\left\{\sum_{j=1}^{n}\sum_{i=1}^{n} x_{sj}^2 + c_{ij_0}x_{si}x_{ji} + c_0 + \sum_{j=1}^{n}\sum_{i=1}^{n} x_{id}^2 + c_{ij_1}x_{id}x_{id} + c_1 + \sum_{k\in V,k\neq s,d}\sum_{i=1}^{n}\sum_{j=1}^{n} c_{jk}x_{kj}^2 + c_{ik}x_{ik}^2 + c_{ij_2}x_{kj}x_{ik}\right\}$$

$$x_{ij}^2 = x_{ij}, \qquad \because x_{ij} = 0 \; or \; 1$$

$$F = \sum_{i,j} c_{ij}x_{ij} + \sum_{ijkl} c_{ijkl}\, x_{ij}x_{kl} + c_0$$

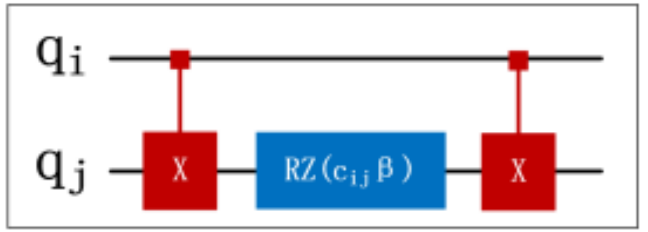
$$O(z) = \sum_{k=0}^{m-1} c_k q_k + \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} c_{ij}q_i q_j$$

Hamiltonian Construction

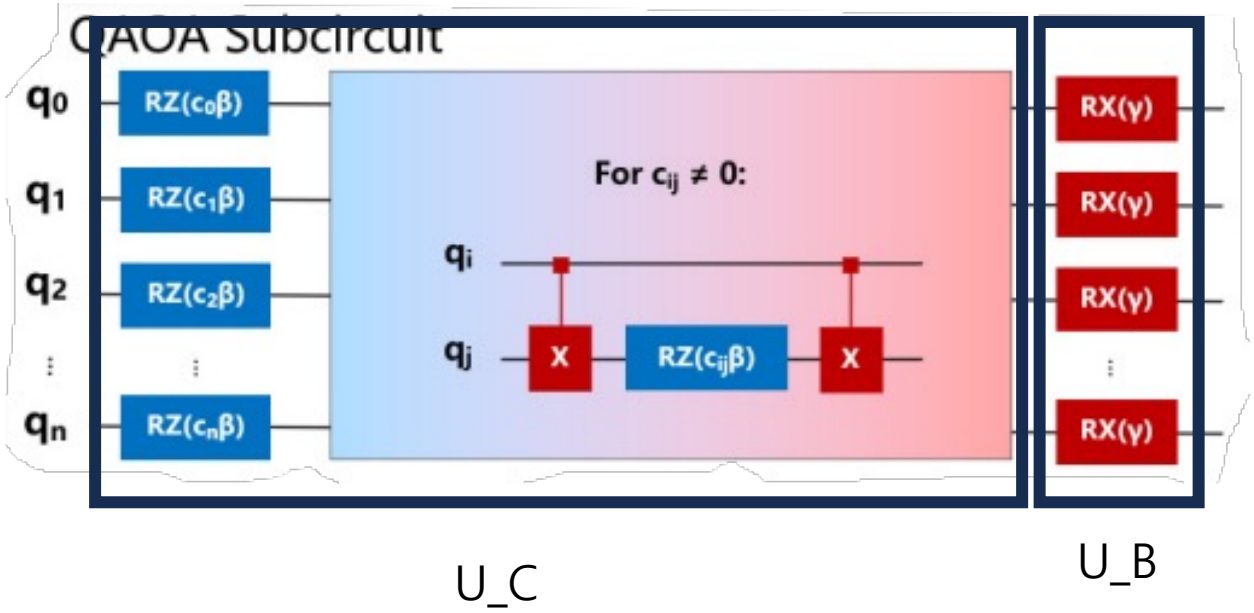
$$H = \sum_{k=0}^{m-1} c_k \hat{\sigma}_k + \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} c_{ij} \hat{\sigma}_i \hat{\sigma}_j$$

$c_k \hat{\sigma}_k \Rightarrow$

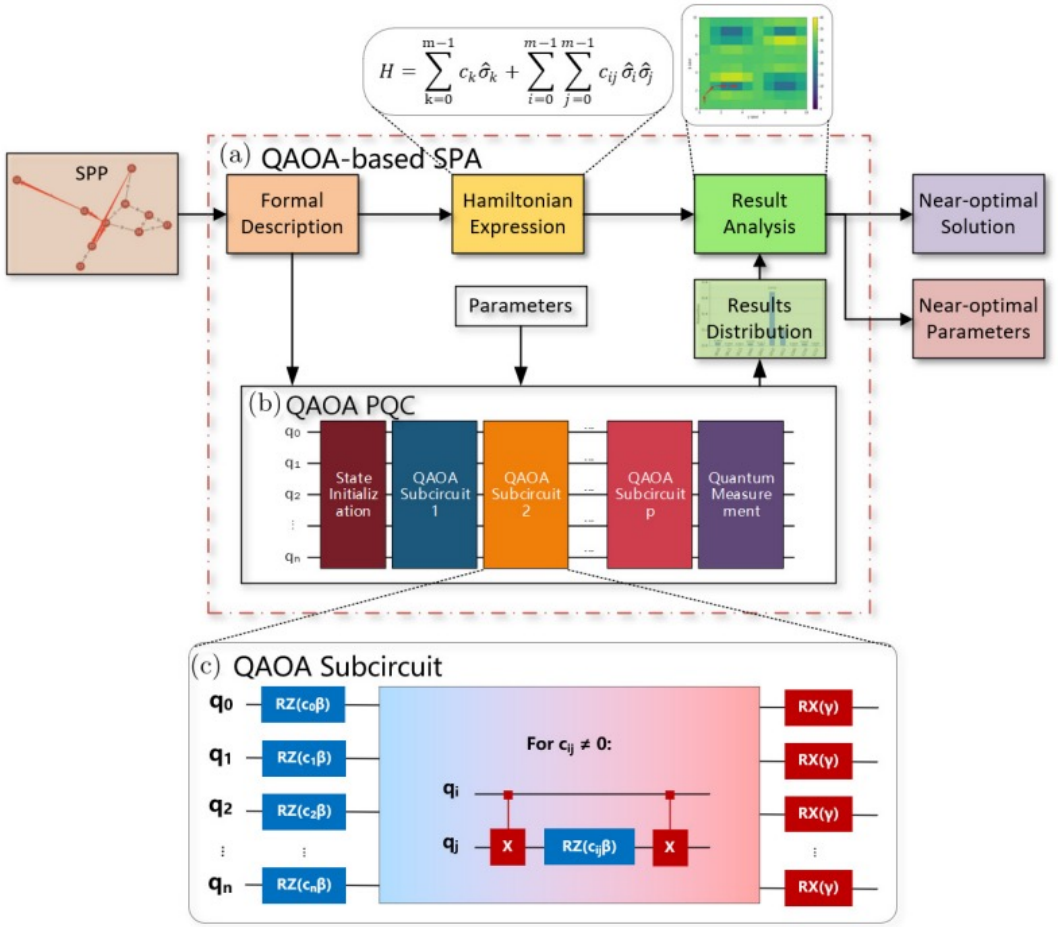
$c_{ij} \hat{\sigma}_i \hat{\sigma}_j \Rightarrow$



QAOA Subcircuit



Shortest Path Problem by QAOA

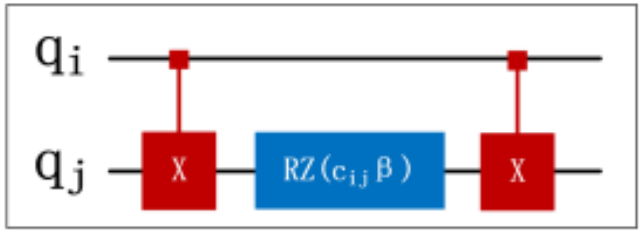


```python
def U_B(beta):
  for wire in range(n_wires):
    qml.RX(2*beta, wires=wire)
```

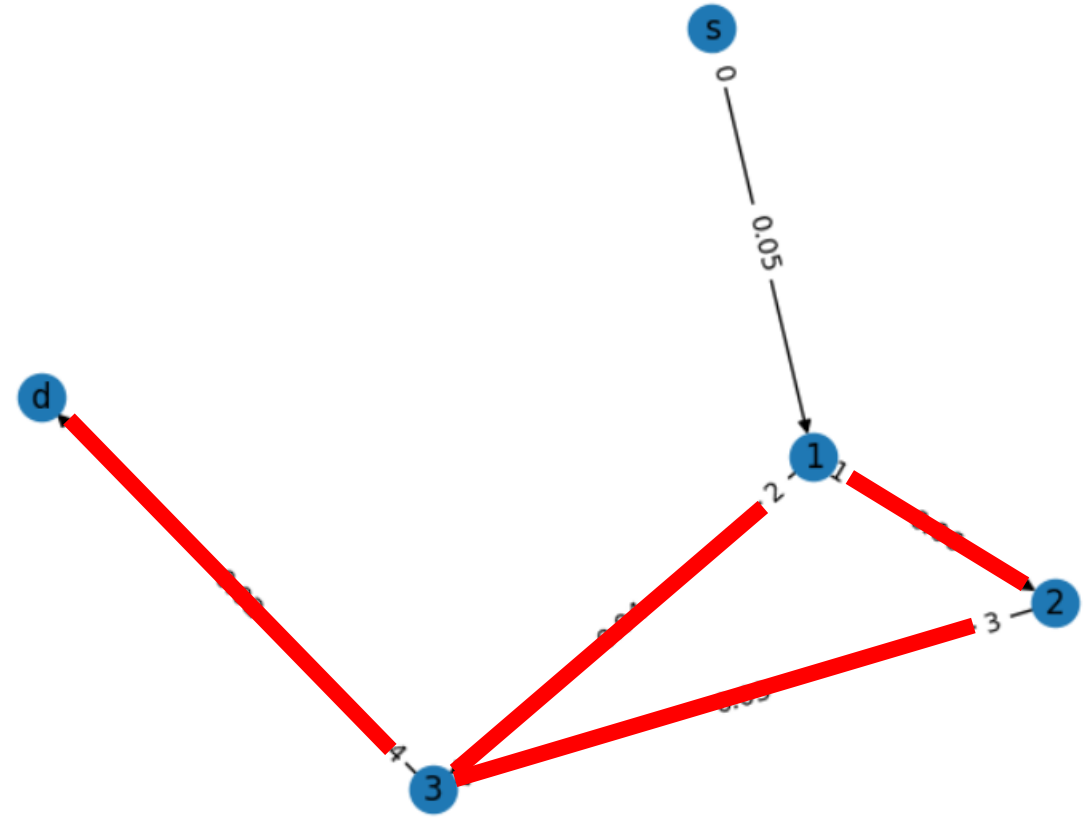

```python
def U_C(gamma):
  for edge in graph.edges:
    if edge[0] == "s" or edge[1] == "d":
      qml.RZ(gamma*edge_labels_weight[edge], wires = edge_labels_number[edge])
    else:
      qml.RZ(gamma*(edge_labels_weight[edge]+2), wires=edge_labels_number[edge])
  for node in graph:
    for edge1 in graph.edges:
      if edge1[1] == node:
        wire1 = edge_labels_number[edge1]
        for edge in graph.edges:
          if edge[0] == node :
            wire2 = edge_labels_number[edge]
            qml.CNOT(wires=[wire1, wire2])
            qml.RZ(gamma*-2, wires=wire2)
            qml.CNOT(wires=[wire1, wire2])
          elif edge[1] == edge1[1] and edge1[0] != edge[0]:
            wire2 = edge_labels_number[edge]
            qml.CNOT(wires=[wire1, wire2])
            qml.RZ(gamma*2, wires=wire2)
            qml.CNOT(wires=[wire1, wire2])
      if edge1[0] == node:
        for edge2 in graph.edges:
          if edge1[0] == edge2[0] and edge1[1] != edge2[1]:
            wire1 = edge_labels_number[edge1]
            wire2 = edge_labels_number[edge2]
            qml.CNOT(wires=[wire1, wire2])
            qml.RZ(gamma*2, wires=wire2)
            qml.CNOT(wires=[wire1, wire2])
```

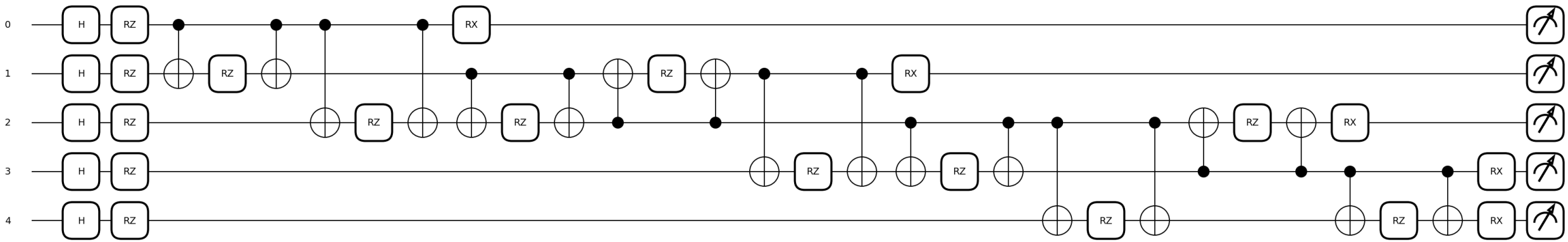
$$c_k \hat{\sigma}_k \Rightarrow$$

$$c_{ij} \hat{\sigma}_i \hat{\sigma}_j \Rightarrow$$



```
for node in graph:
  for edge1 in graph.edges:
    if edge1[1] == node:
      wire1 = edge_labels_number[edge1]
      for edge in graph.edges:
        if edge[0] == node :
          wire2 = edge_labels_number[edge]
          qml.CNOT(wires=[wire1, wire2])
          qml.RZ(gamma*-2, wires=wire2)
          qml.CNOT(wires=[wire1, wire2])
        elif edge[1] == edge1[1] and edge1[0] != edge[0]:
          wire2 = edge_labels_number[edge]
          qml.CNOT(wires=[wire1, wire2])
          qml.RZ(gamma*2, wires=wire2)
          qml.CNOT(wires=[wire1, wire2])
    if edge1[0] == node:
      for edge2 in graph.edges:
        if edge1[0] == edge2[0] and edge1[1] != edge2[1]:
          wire1 = edge_labels_number[edge1]
          wire2 = edge_labels_number[edge2]
          qml.CNOT(wires=[wire1, wire2])
          qml.RZ(gamma*2, wires=wire2)
          qml.CNOT(wires=[wire1, wire2])
```

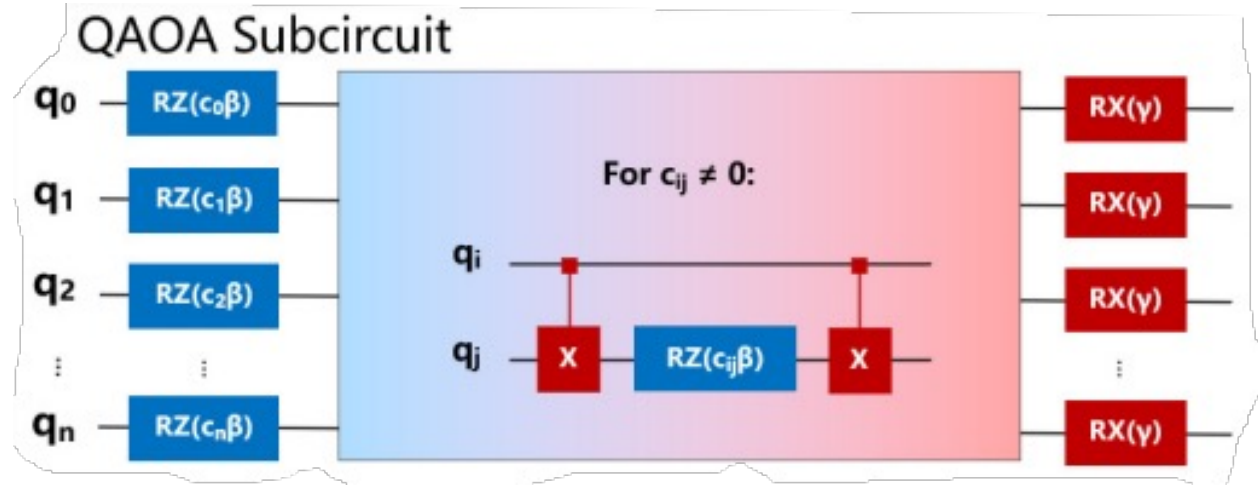


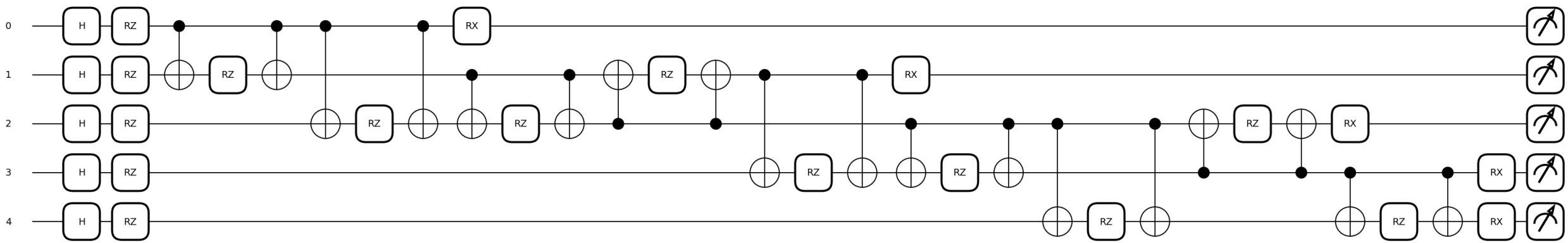


```python
dev = qml.device("default.qubit", wires=n_wires, shots=1)

@qml.qnode(dev)
def circuit(gammas, betas, edge=None, n_layers=1):
  for wire in range(n_wires):
    qml.Hadamard(wires=wire)
  for i in range(n_layers):
    U_C((gammas[i]))
    U_B((betas[i]))
  if edge is None:
    return qml.sample()
  H = (qml.PauliZ(edge_labels_number[edge]) + 1) / 2

  return qml.expval(H)
```





```
def objective(params):
  gammas = params[0]
  betas = params[1]
  obj = 0

  xsi = 0
  xid = 0
  temp = 0
  penalty = 0
  expval = 0

  for edge in graph.edges:
    expval = circuit(gammas, betas, edge=edge, n_layers=n_layers)
    obj += edge_labels_weight[edge] * expval
    if edge[0] == "s":
      xsi += expval
    if edge[1] == "d":
      xid += expval
  for node in graph:
    if node != "s" and node != "d":
      for edge in graph.edges:
        expval =  circuit(gammas, betas, edge=edge, n_layers=n_layers)
        if edge[0] == node:
          temp += expval
        elif edge[1] == node:
          temp -= expval
      penalty += temp * temp
      temp = 0
  penalty += (xsi - 1)*(xsi - 1) + (xid - 1) * (xid - 1)
  obj += penalty * 10
  return obj
```

$$F = C + M\left(\sum_{j=1}^{n} x_{sj} - 1\right)^2 + M\left(\sum_{i=1}^{n} x_{id} - 1\right)^2 + M\sum_{k\in V, k\neq s,d}\left(\sum_{i=1}^{n} x_{ik} - \sum_{j=1}^{n} x_{kj}\right)^2$$

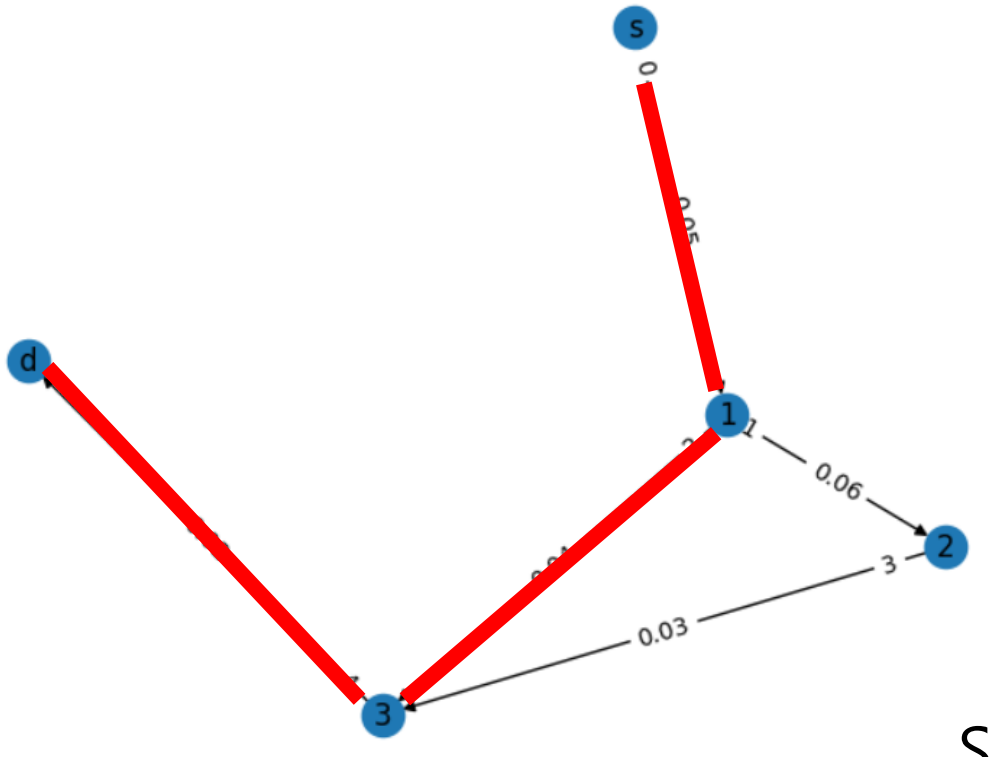
$$C = \sum_{i,j} w_{ij} x_{ij}$$

```
  init_params = 0.01 * np.random.rand(2, n_layers, requires_grad=True)
```

```
  opt = qml.AdamOptimizer(stepsize=0.5)
  params = init_params
  steps = 40
  for i in range(steps):
    params = opt.step(objective, params)
    if (i +1) % 5 == 0:
      print("Objective after step {:5d}: {: .7f}".format(i+1, objective(params)))
  bit_strings = []
  n_samples = 400
  for i in range(0, n_samples):
    bit_strings.append(bitstring_to_int(circuit(params[0], params[1], edge=None, n_layers=n_layers)))
  counts = np.bincount(np.array(bit_strings))
  most_freq_bit_string = np.argmax(counts)
  print("Optimized (gamma, beta) vectors:₩n{}".format(params[:, :n_layers]))
  print("Most frequently sampled bit string is: {:05b}".format(most_freq_bit_string))
  return objective(params), bit_strings

bitstrings1 = qaoa_SPP(n_layers=1)[1]
bitstrings2 = qaoa_SPP(n_layers=2)[1]
bitstrings3 = qaoa_SPP(n_layers=3)[1]
bitstrings4 = qaoa_SPP(n_layers=4)[1]
```

Results

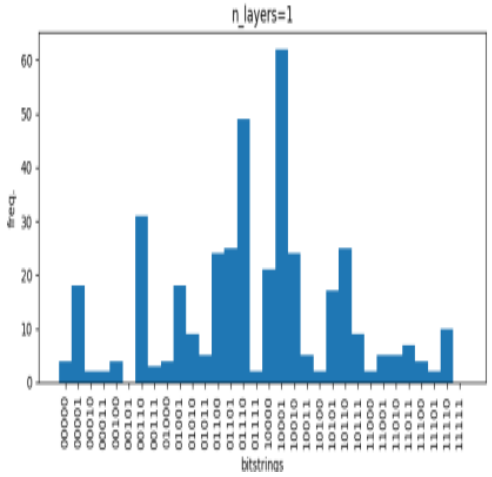


Shortest path : 10101

Results

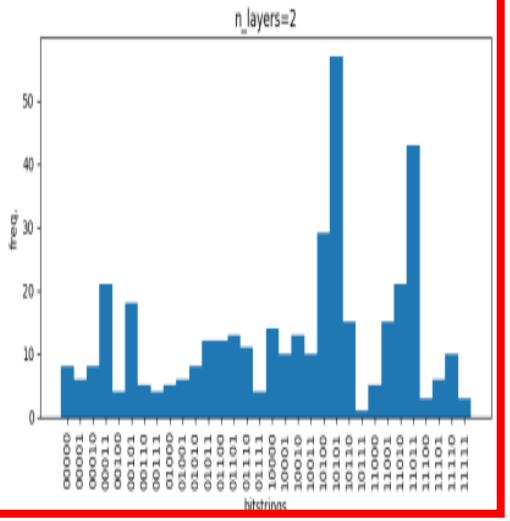
p=1

```
Objective after step     5:  50.0300000
Objective after step    10:  40.0600000
Objective after step    15:  20.0700000
Objective after step    20:  40.1300000
Objective after step    25:  80.1000000
Objective after step    30:  20.1600000
Objective after step    35:  30.0900000
Objective after step    40:  40.0500000
Optimized (gamma, beta) vectors:
[[-0.28916815]
 [-0.44225358]]
Most frequently sampled bit string is: 10001
```



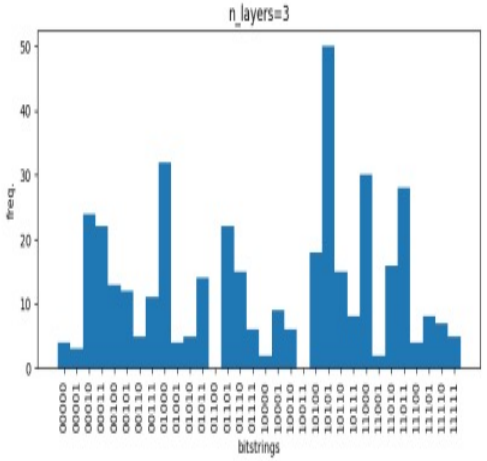
p=2

```
Objective after step     5:  50.0500000
Objective after step    10:  30.0200000
Objective after step    15:  50.0900000
Objective after step    20:  10.2000000
Objective after step    25:  30.1000000
Objective after step    30:  50.1700000
Objective after step    35:  60.0700000
Objective after step    40:  30.1200000
Optimized (gamma, beta) vectors:
[[-1.81981465  3.20761915]
 [ 2.27639114  2.23920504]]
Most frequently sampled bit string is: 10101
```



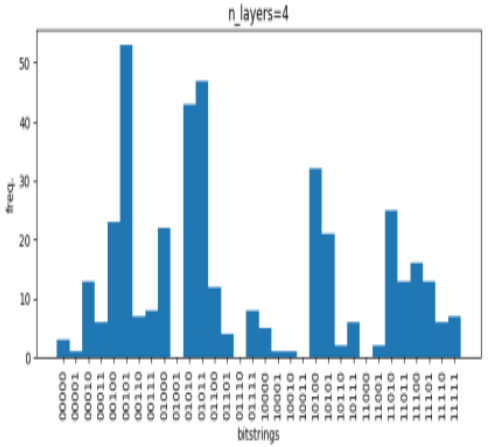
p=3

```
Objective after step     5:  30.0900000
Objective after step    10:  40.2000000
Objective after step    15:  20.1800000
Objective after step    20:  40.0700000
Objective after step    25:  40.0700000
Objective after step    30:  30.0900000
Objective after step    35:  50.1300000
Objective after step    40:  40.2000000
Optimized (gamma, beta) vectors:
[[ 5.08037923 -0.05139518 -1.49106949]
 [ 4.05721858  3.42189422  3.78406485]]
Most frequently sampled bit string is: 10101
```

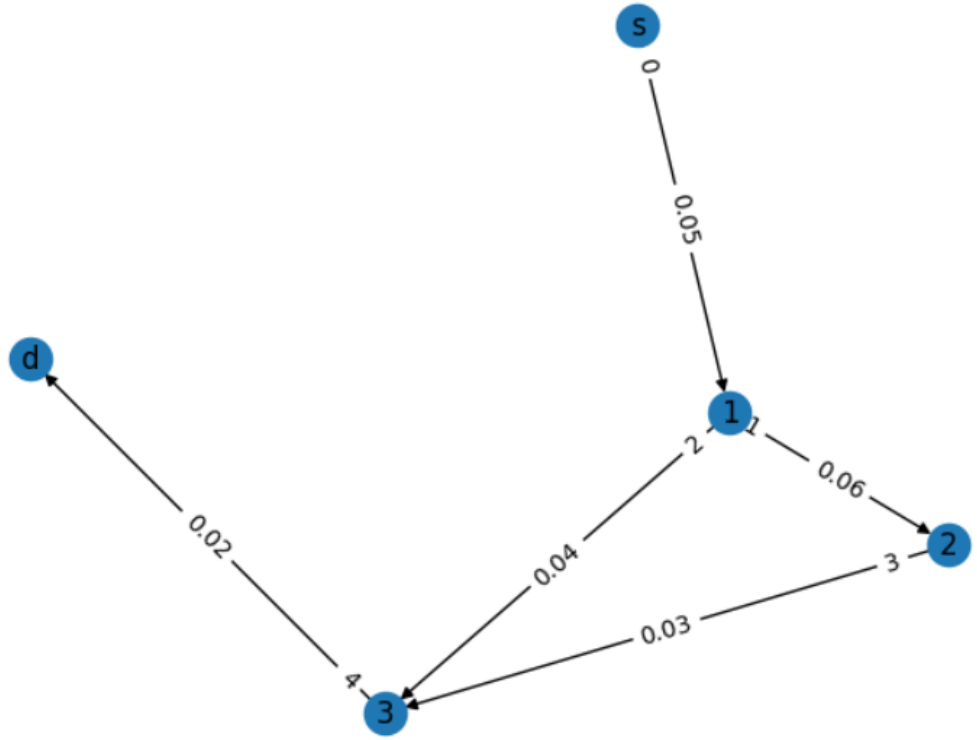


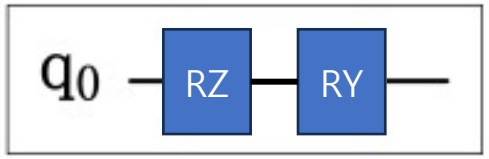
p=4

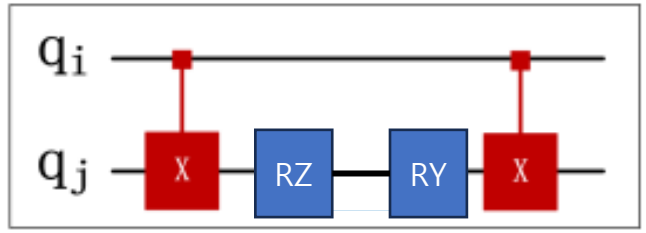
```
Objective after step     5:  10.1100000
Objective after step    10:  10.1200000
Objective after step    15:  20.0700000
Objective after step    20:  10.1000000
Objective after step    25:  30.1500000
Objective after step    30:  30.0800000
Objective after step    35:  30.1500000
Objective after step    40:  20.1200000
Optimized (gamma, beta) vectors:
[[ 0.07337043 -7.45977662 -1.95772262  1.62214439]
 [-1.95763738 -1.28216974  2.40700169  0.8956126 ]]
Most frequently sampled bit string is: 00101
```



Results



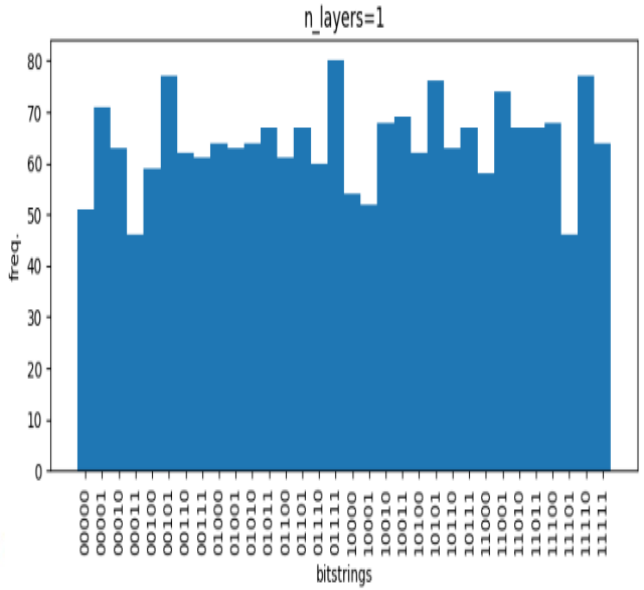
$c_k \hat{\sigma}_k \Rightarrow$ 

$c_{ij} \hat{\sigma}_i \hat{\sigma}_j \Rightarrow$ 

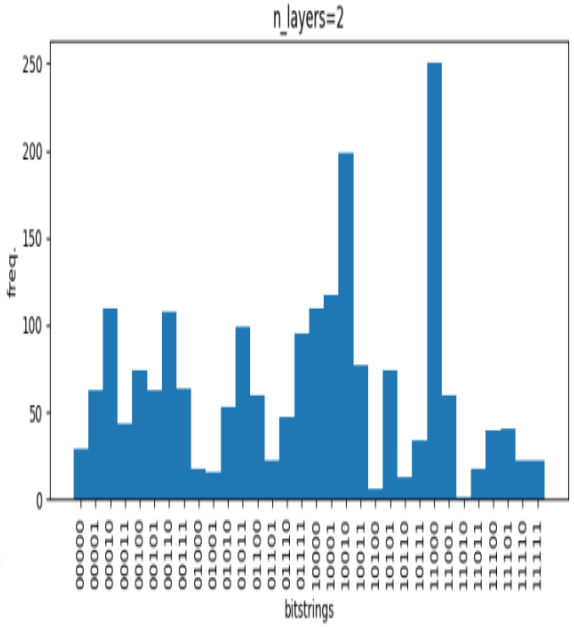
Shortest path : 10101

Results

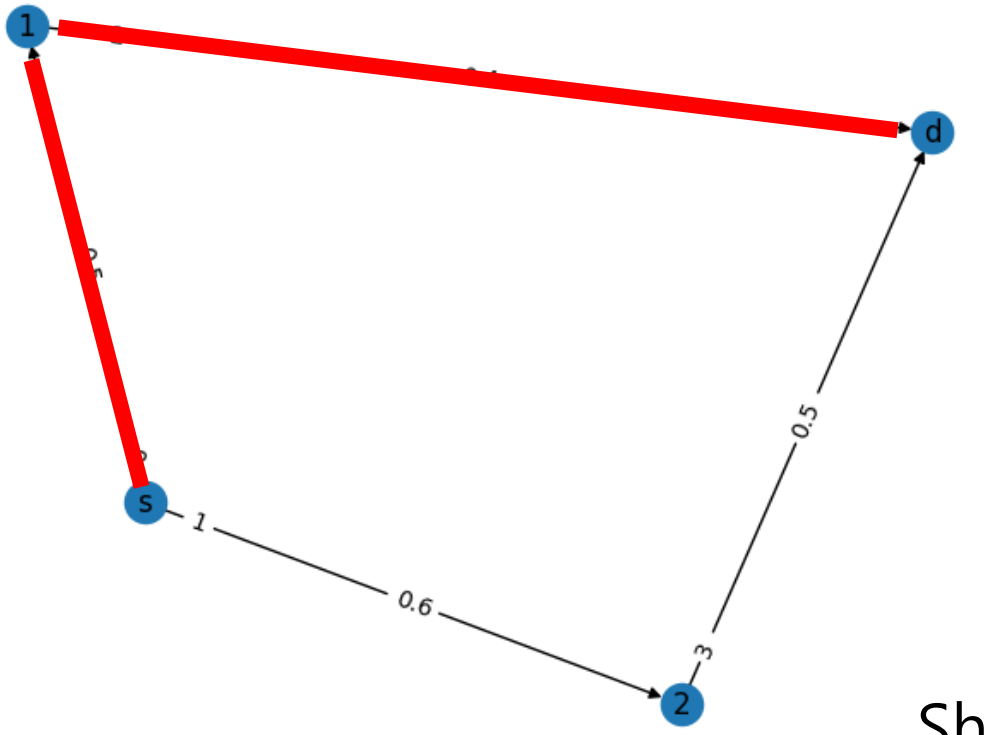
```
p=1
Objective after step     5:  15.1400000
Objective after step    10:  10.1100000
Objective after step    15:  5.1300000
Objective after step    20:  10.0500000
Objective after step    25:  15.1200000
Objective after step    30:  15.0600000
Objective after step    35:  30.1200000
Objective after step    40:  15.1500000
Objective after step    45:  20.0400000
Objective after step    50:  10.0800000
Optimized (gamma, beta) vectors:
[[6.27641788]
 [5.12910492]]
Most frequently sampled bit string is: 011
```




```
p=2
Objective after step     5:  5.1100000
Objective after step    10:  10.0200000
Objective after step    15:  5.0500000
Objective after step    20:  30.1500000
Objective after step    25:  10.1400000
Objective after step    30:  10.0800000
Objective after step    35:  5.0900000
Objective after step    40:  10.1400000
Objective after step    45:  25.1300000
Objective after step    50:  35.0900000
Optimized (gamma, beta) vectors:
[[ 5.76033383  0.66634441]
 [ 1.07823201 12.41579433]]
Most frequently sampled bit string is: 11000
```



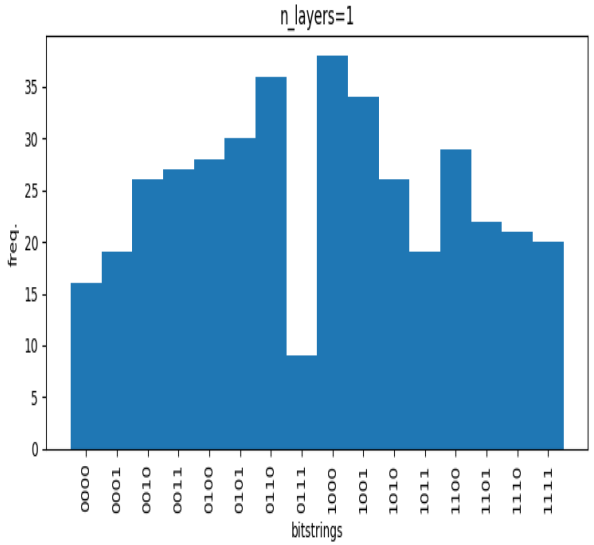
Results



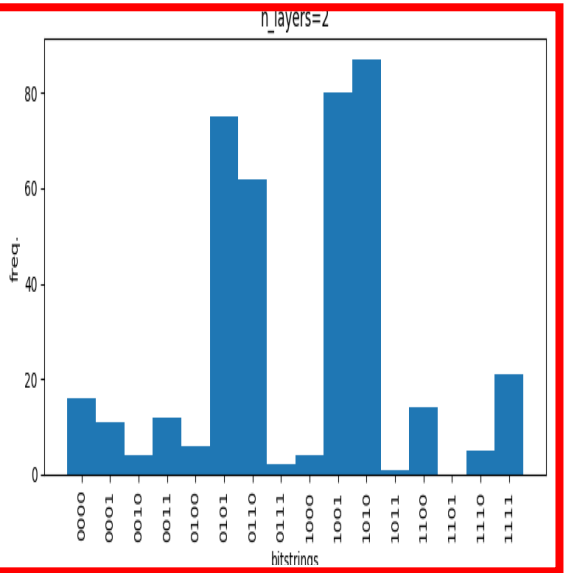
Shortest path : 1010

Results

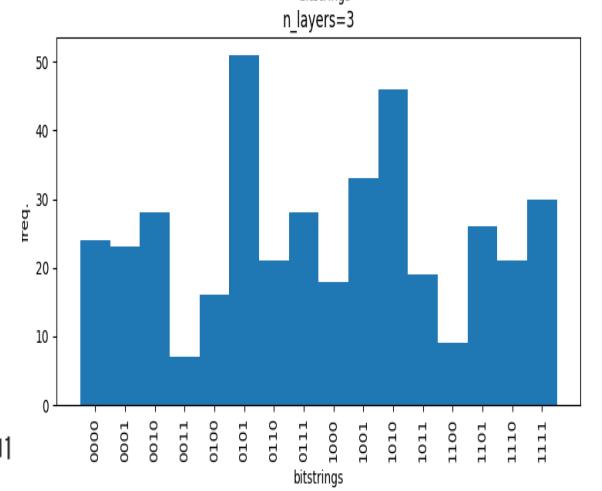
p=1
```
Objective after step     5:  11.0000000
Objective after step    10:  20.6000000
Objective after step    15:  30.9000000
Objective after step    20:  11.5000000
Objective after step    25:  10.9000000
Objective after step    30:  30.5000000
Objective after step    35:  31.5000000
Objective after step    40:  21.0000000
Optimized (gamma, beta) vectors:
[[ 0.52091047]
 [-0.05669381]]
Most frequently sampled bit string is: 1000
```



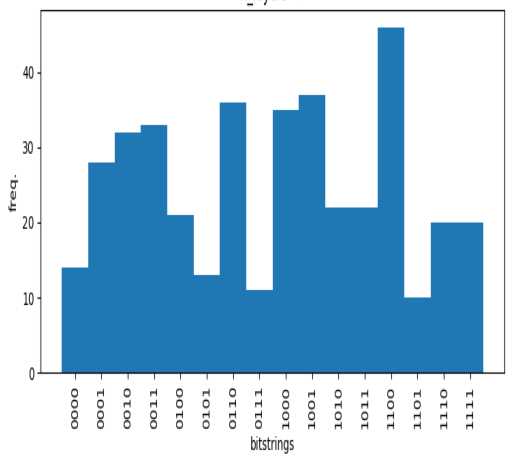
p=2
```
Objective after step     5:  21.5000000
Objective after step    10:  11.0000000
Objective after step    15:  20.6000000
Objective after step    20:  20.5000000
Objective after step    25:  20.5000000
Objective after step    30:  30.9000000
Objective after step    35:  11.4000000
Objective after step    40:  30.0000000
Optimized (gamma, beta) vectors:
[[ 0.41223602  0.42149636]
 [-0.31568882  0.0393566 ]]
Most frequently sampled bit string is: 1010
```



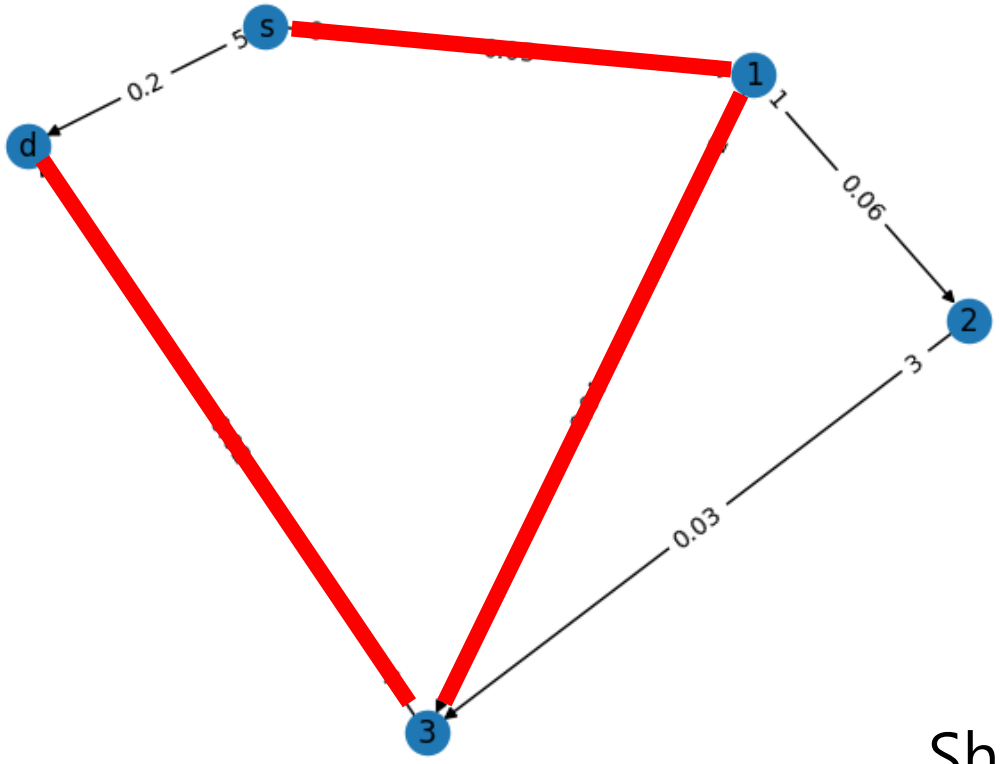
p=3
```
Objective after step     5:  31.5000000
Objective after step    10:  20.4000000
Objective after step    15:  11.4000000
Objective after step    20:  20.4000000
Objective after step    25:  21.4000000
Objective after step    30:  20.5000000
Objective after step    35:  20.5000000
Objective after step    40:  21.1000000
Optimized (gamma, beta) vectors:
[[ 0.00699484 -0.19066262  0.43694152]
 [ 0.02768758  0.2933984   0.14498354]]
Most frequently sampled bit string is: 0101
```



p=4
```
Objective after step     5:  31.1000000
Objective after step    10:  32.0000000
Objective after step    15:  20.5000000
Objective after step    20:  20.9000000
Objective after step    25:  21.5000000
Objective after step    30:  21.6000000
Objective after step    35:  10.9000000
Objective after step    40:  31.5000000
Optimized (gamma, beta) vectors:
[[-0.02234741 -0.05214028 -0.01158323 -0.42574265]
 [-0.16743896 -0.20624738 -0.18893929  0.1112439 ]]
Most frequently sampled bit string is: 1100
```



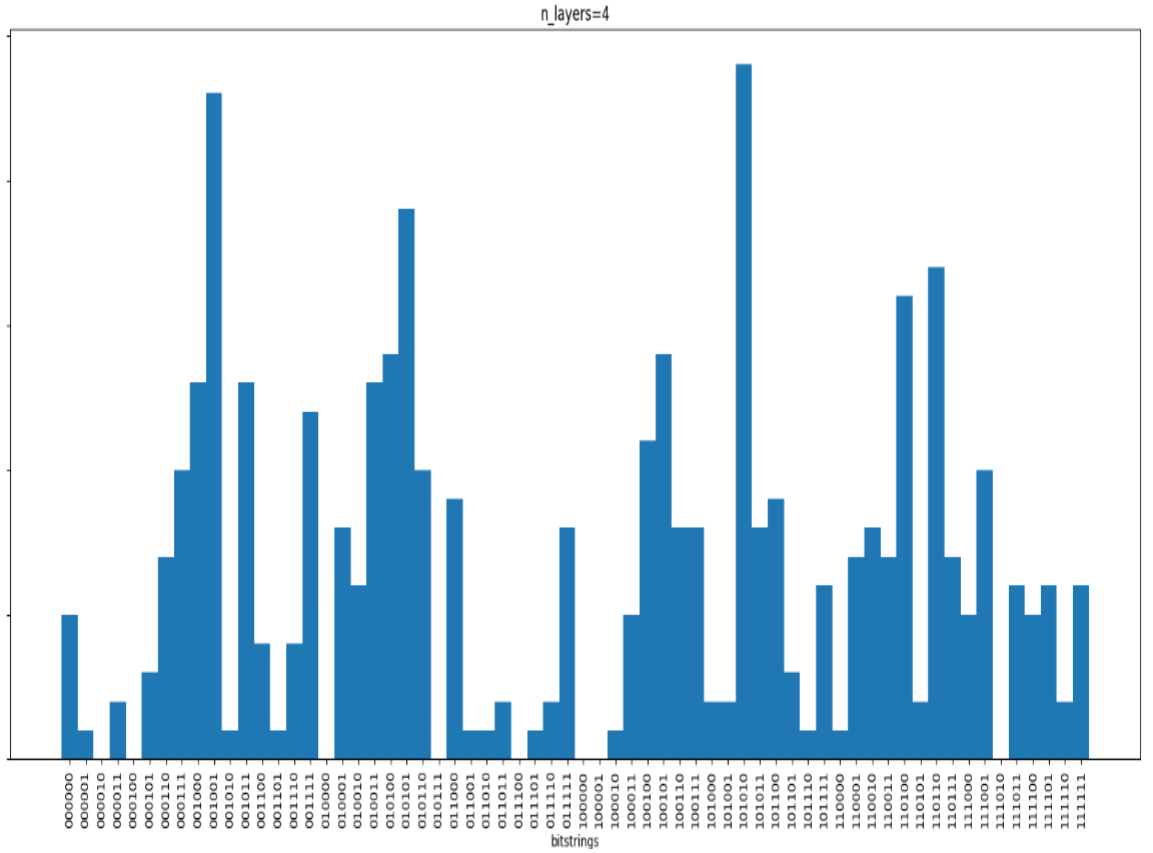
Results



Shortest path : 101010

Results

```
p=4
Objective after step     5:  20.0900000
Objective after step    10:  60.3500000
Objective after step    15:  60.1200000
Objective after step    20:  30.2900000
Objective after step    25:  30.3500000
Objective after step    30:  50.3700000
Objective after step    35:  60.0900000
Objective after step    40:  70.3500000
Optimized (gamma, beta) vectors:
[[-0.45834695  0.17031555  0.21215723 -0.08853176]
 [ 0.32241875  0.22195918 -0.11937346 -0.01191828]]
Most frequently sampled bit string is: 101010
```



감사합니다

Reference

Zhiqiang Fan, Jinchen Xu, Guoqiang Shu, Xiaodong Ding, Hang Lian, and Zheng Shan. 2023. Solving the Shortest Path Problem with QAOA, SPIN Vol. 13, No. 1 (2023) 2350002